IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF VIRGINIA
RICHMOND DIVISION

THE TRUSTEES OF COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK, *Plaintiff*	Civil Action No. 3:13-cv-00808-JRS
vs.	**JURY TRIAL DEMANDED**
SYMANTEC CORPORATION, *Defendant*	

SECOND DECLARATION OF PROFESSOR DOUGLAS C. SZAJDA

1. I have been asked to consider the Declaration of Dr. Trent Jaeger and his discussion of the concept of a byte sequence feature.

2. My qualifications are recited in the first declaration I have submitted in this matter.

3. In preparing this declaration I have considered the legal standards recited in paragraphs 9–11 of Dr. Jaeger's declaration. I also note that the conclusions in my first declaration remain unchanged when applying Dr. Jaeger's standards.

<u>Byte Sequence Features</u>

4. Dr. Jaeger maintains that a person of ordinary skill in the art would understand byte sequence features as only encompassing machine code instructions and not a string of bytes representative of resources. My first declaration at paragraphs 32 to 44 explained that a byte sequence feature would be understood by a person of ordinary skill in the art of the '544/'907 patents as a property or attribute of a sequence of bytes which may take on a set of values. A person of ordinary skill in the art would understand with reasonable certainty that byte sequence features include machine code instructions as well as other information that may be extracted from a file. A person of ordinary skill in the art also would understand that byte sequence features extracted from a file may include byte strings representative of resources referenced by the executable. The latter is thus an example of a byte sequence feature. I will not repeat the analysis in my first declaration. Instead, I will discuss the examples of different types of byte sequence features in the specification.

5. The basic unit of information in computers is the bit—a 1 or a 0. And a collection of bits (typically 8) is a byte. There are many ways to display sequences (or "strings") of bytes that are relevant to this case. One type of notation is hexadecimal format.

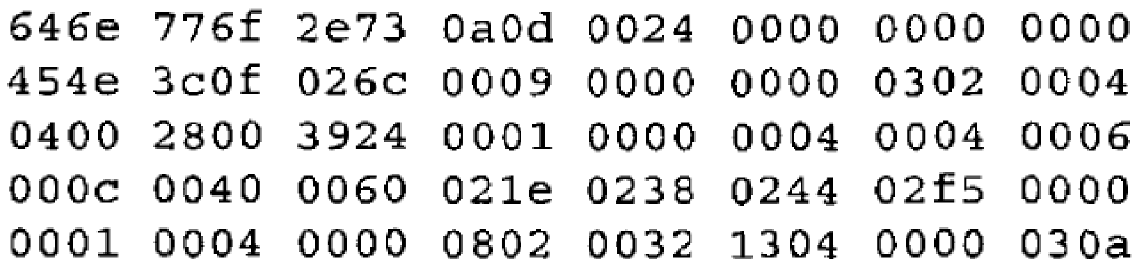
Another is ASCII. The ’544/’907 patents depict byte sequences in both hexadecimal and ASCII format.

6. Hexadecimal format represents a byte as two hexadecimal digits. For example, the digits “64” in hexadecimal format represent the byte “01100100” (displayed in binary notation). See Ex. DD [http://www.neurophys.wisc.edu/comp/docs/ascii/#table].

7. ASCII means American Standard Code for Information Exchange II, an international standard for representing a byte as different letters, numbers or punctuation marks. Ex. EE [http://sliderule.mraiow.com/w/images/7/73/ASCII.pdf]. So, for example, the letter “A” represents the byte “01000001” (displayed in binary notation). See Ex. DD [http://www.neurophys.wisc.edu/comp/docs/ascii/#table].

8. Attached Exhibit DD [http://www.neurophys.wisc.edu/comp/docs/ascii/#table] is a helpful chart showing how bytes and byte sequences can be represented in different notations, including hexadecimal, binary, and ASCII.

9. Figure 2 from the ’544/’322 patents is a sequence, or string, of bytes in hexadecimal format, created from the program “hexdump.”



```
646e 776f 2e73 0a0d 0024 0000 0000 0000
454e 3c0f 026c 0009 0000 0000 0302 0004
0400 2800 3924 0001 0000 0004 0004 0006
000c 0040 0060 021e 0238 0244 02f5 0000
0001 0004 0000 0802 0032 1304 0000 030a
```

FIG. 2

10. Figure 3 from the ’544/’322 patents is an example of a sequence, or string, of bytes representative of resources accessed by an executable.

$$\neg advapi32 \wedge avicap32 \wedge \cdots \wedge winmm \wedge \neg wsock32$$

FIG. 3

11. Figures 2 and 3 both show byte sequence features. They are simply different ways of displaying the byte sequence features extracted from the executable. Figure 2 uses hexadecimal format to represent the extracted byte sequence features. Figure 3 uses ASCII format to depict the extracted byte sequence features. Further, each form of representing byte sequence features is easily convertible to the other. ASCII strings (as shown in Figure 3) could be represented in hexadecimal format (as shown in Figure 2), and vice versa. See Ex DD [http://www.neurophys.wisc.edu/comp/docs/ascii/#table]. This is similar to how fonts work. Formatting a piece of text in Times New Roman font versus Courier font does not change which characters form the text. Instead, it determines the visual presentation of the text.

12. I note also that that Dr. Jaeger ignores the fact that the hexdump program generates hexadecimal strings based not only on machine code instructions, but on anything in a program. This can be illustrated by considering the "Portable Executable (PE)" header section of an executable. PE formatting is the standardized formatting used for executables in the Windows environment. Programs in PE format contain multiple different sections. The patents describe the structure of PE format programs as follows: "In PE, or Common Object File Format (COFF), program headers are composed of a COFF header, an Optional header, at MS-DOS stub, and a file signature. All of the information about the binary is obtained from the program." Ex. C, '544 patent at 6:48–54.

13. Below is an example of a Window PE file, including multiple header sections and image pages:

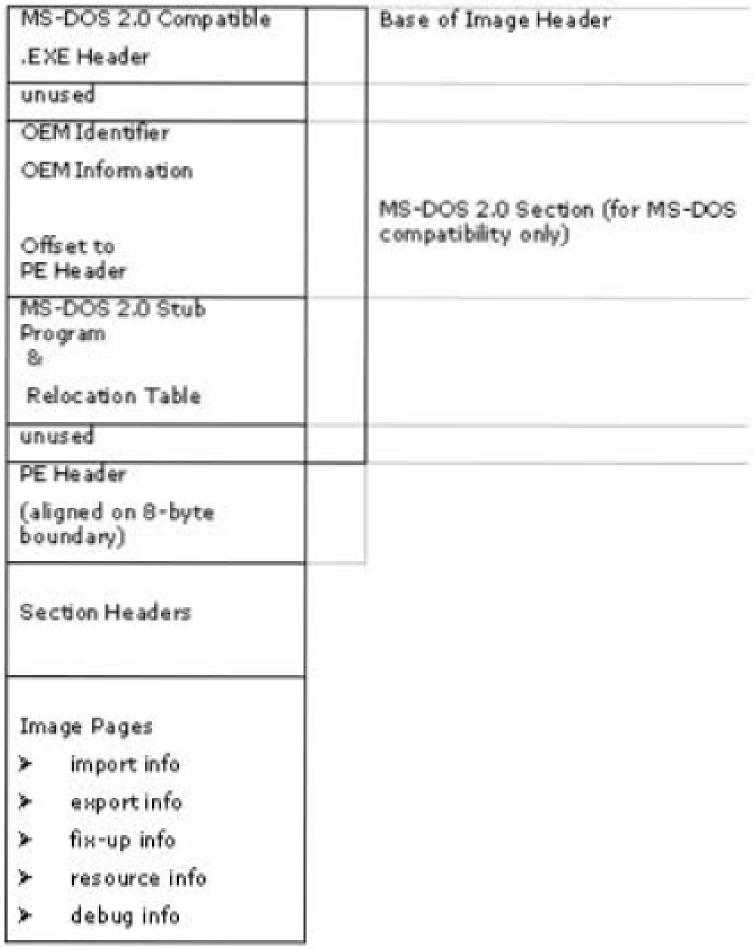
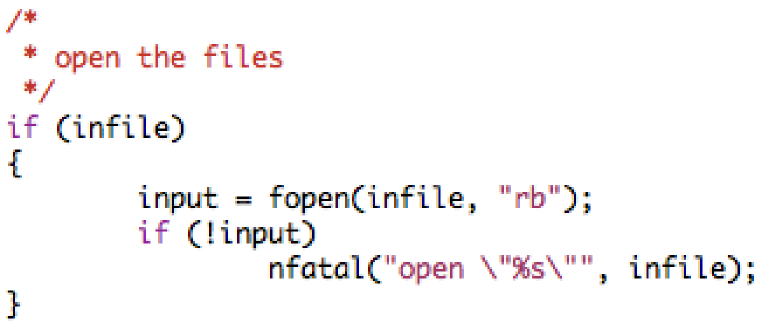


Figure 1. Typical 32-Bit Portable .EXE File Layout

Exhibit H at 9.

14. When hexdump is used on a program in PE format, it will create a hexadecimal string based on the information in the entire PE file, not just the machine code instructions in isolated sections. For example, as depicted above, the MS-DOS 2.0 Compatible .EXE Header, the PE Header, and the Image Pages will all be processed by hexdump and included in the hexadecimal string that is output by hexdump.

15. Below is the code from the version of hexdump (circa 1999, cited in the ’544 patent) that allows the program to convert everything in the executable into hexadecimal:



```
/*
 * open the files
 */
if (infile)
{
        input = fopen(infile, "rb");
        if (!input)
                nfatal("open \"%s\"", infile);
}
```

The line that contains the call to fopen(), with second parameter “rb” opens a binary stream to the specified input file.

<u>GNU Strings</u>

16. I have also been asked to provide a short background on the “GNU strings” program referenced in the specification. When writing computer code, a programmer will often use ASCII characters. A subset of ASCII characters are considered printable characters. These characters include a number of punctuation marks, the numbers 0-9 and lower case and capitalized versions of the letters A–Z. When programs include multiple ASCII printable characters together it is called a printable string. GNU strings is a program that will read through an executable and extract all of the printable strings.

17. Table 1 of the ’544/’907 patents provides an example of printable strings from a program:

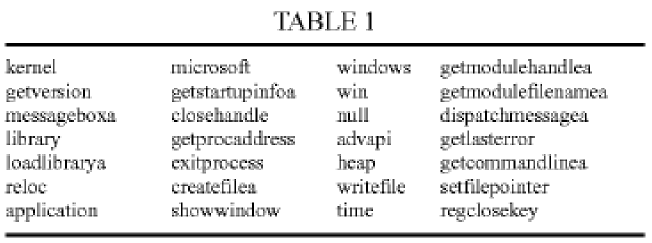


TABLE 1

kernel	microsoft	windows	getmodulehandlea
getversion	getstartupinfoa	win	getmodulefilenamea
messageboxa	closehandle	null	dispatchmessagea
library	getprocaddress	advapi	getlasterror
loadlibrarya	exitprocess	heap	getcommandlinea
reloc	createfilea	writefile	setfilepointer
application	showwindow	time	regclosekey

18. None of the words listed in Table 1 is a machine code instruction. Instead, they are printable strings that are used as data in various contexts in the program. For example, some of the printable strings are displayed to the user as part of the program's interface. There are also strings that correspond to the parameters passed as function arguments. Other strings are function names that are being used as data—for example, as references to external functions that are imported by the program. Although the words may refer to a function, the words themselves are not machine code instructions and could not be executed by a central processing unit.

<u>To a Person of Ordinary Skill in the Art, Byte Strings Representative of Resources are Examples of Byte Sequence Features in the '544/'907 patents</u>

19. The Summary section of the specification for the '544/'907 patents reasonably apprises a person of ordinary skill in the art that more than machine code instructions can be included in byte sequence features. Instead, it makes explicit that byte strings representative of resources can be byte sequence features. Ex. C, '544 patent at 3:30–40. This is also made explicit in the Detailed Description of Exemplary Embodiments. This section discusses, for example, Figure 1, which refers to Step 20 "Extract Features from Data," which would be reasonably understood as the extraction of byte sequence features. The output of the hexdump program, and byte strings representative of resources, are described as alternative examples of information generated from Step 20, which is to say information that can populate byte sequence features. Column 5, line 56 to column 6, line 6 introduces the concept of byte sequence feature. Column 6, line 7 to column 7, line 63 are concrete examples of the types of information that can populate the byte sequence features. This includes byte strings representative of resources referenced.

20. Nothing in the Provisional Patent Application No. 60/308,622 alters what a person of ordinary skill in the art would reasonably understand about the patents. The provisional application is an academic article. A person of ordinary skill in the art would not understand it as altering the clear statements in the specification. To the contrary, a person of ordinary skill would understand the provisional application as teaching that all information in an executable, not just machine code instructions, can be used as byte sequence features. Ex. 2, Provisional Application No. 60/308,622, at 5 ("[A]nalyzing ***the entire binary*** gives more information for non-PE format executables than the strings method.") (emphasis added).

21. The same holds for claim 28 of the '544 patent. It does not alter what a person of ordinary skill would reasonably understand about the patents. A person of ordinary skill in the art would understand this claim as teaching a system with a feature extractor that is capable of extracting a byte sequence feature, but not just any byte sequence feature. It must be capable of extracting byte sequence features that are representative of resources referenced by the executable email attachment.

<u>Emulator</u>

22. I have also been asked to consider the Declaration of Dr. Richard Ford and his discussion of emulators. Dr. Ford maintains that emulator, as that term is used in the '115/'322 patents, requires the use of a simulation. Ford Decl., ¶ 12. I discuss at length that simulation is not a required feature of emulator as used in the patents at paragraphs 79–85, and paragraphs 91–96 of my first declaration. I will not repeat the analysis presented in my first declaration but I will provide additional examples of emulators described in the patents that do not use what Dr. Ford describes as "fake" analysis. These embodiments analyze the actual code of the executable. The patent specification states:

> In some embodiments, the instruction-level emulator may be linked with the application in advance. Alternatively, in response to a detected failure, the instruction-level emulator may be compiled in the code. In another suitable embodiment, the instruction-level emulator may be invoked in a manner similar to a modem debugger when a particular pro- gram instruction is executed. This can take advantage of breakpoint registers and/or other program debugging facilities that the system processor and architecture possess, or it can be a pure-software approach. 14:6-14.

This passage describes to a person of ordinary skill examples of emulation that analyze program instructions dynamically as a program runs. In one embodiment, the specification teaches the insertion of an instruction-level emulator into the actual executable so that as the executable runs the emulation occurs within the executable run: "[a]lternatively, in response to a detected failure, the instruction-level emulator may be compiled in the code." In another embodiment, the emulation function described in the patent is performed in a manner akin to a debugger which allows for the code to be monitored and faults to be avoided. Debugging, of course, allows for the real-time analysis of code as it runs.

23. Attached as Exhibit HH is portions of the user manual for the graphic GNU debugger, one of the most widely used debuggers in the world. As this section makes clear, debuggers work on the program code: "The purpose of a debugger such as GDB is to allow you to see what is going on "inside" another program while it executes—or what another program was doing at the moment it crashed." Ex. HH [https://sourceware.org/gdb/current/onlinedocs/gdb/Summary.html#Summary].

24. I declare under penalty of perjury that the foregoing is true and correct to the best of my knowledge.

Executed August 28, 2014, in Richmond, Virginia.

[illegible]

Professor Douglas C. Szajda